RHmalloc:

A Very Large, Highly Concurrent Dynamic Memory Manager

Thesis submitted for the degree of

Doctor of Philosophy

Raymond John Huetter

Faculty of Information Technology

University of Technology, Sydney

February 2005

for Ziggi and Jazaan

believe in yourselves

Certificate of Authorship/Originality

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Raymond John Huetter
Faculty of Information Technology
University of Technology, Sydney
February 2005

Abstract

Dynamic memory management (DMM) is a fundamental aspect of computing, directly affecting the capability, performance and reliability of virtually every system in existence today. Yet oddly, the fifty year research into DMM has not taken memory capacity into account, having fallen significantly behind hardware trends. Comparatively little research work on scalable DMM has been conducted – of the order of ten papers exist on this topic – all of which focus on CPU scalability only; the largest heap reported in the literature to date is 600MB. By contrast, symmetric multi-processor (SMP) machines with terabytes of memory are now commercially available.

The contribution of our research is the formal exploration, design, construction and proof of a general purpose, high performance dynamic memory manager which scales indefinitely with respect to both CPU and memory – one that can predictably manage a heap of arbitrary size, on any SMP machine with an arbitrary number of CPU's, without *a priori* knowledge.

We begin by recognizing the scattered, inconsistency of the literature surrounding this topic. Firstly, to ensure clarity, we present a simplified introduction, followed by a catalog of the fundamental techniques. We discuss the melting pot of engineering tradeoffs, so as to establish a sound basis to tackle the issue at hand – large scale DMM. We review both the history and state of the art, from which significant insight into this topic is to be found. We then explore the problem space and suggest a workable solution.

Our proposal, known as RHmalloc, is based on the novel perspective that, a highly scalable heap can be viewed as, an unbounded set of finite-sized sub-heaps, where each sub-heap maybe concurrently shared by any number of threads; such that a suitable sub-heap can be found in O(1) time, and an allocation from a suitable sub-heap is also O(1).

Testing the design properties of RHmalloc, we show by extrapolation that, RHmalloc will scale to at least 1,024 CPU's and 1PB; and we theoretically prove that DMM scales indefinitely with respect to both CPU and memory.

Most importantly, the approach for the scalability proof alludes to a general analysis and design technique for systems of this nature.

Outline of this Dissertation

It is generally considered that the literature on dynamic memory management is inconsistent and scattered [WJNB1995]. According to Wilson et al. we know little more about DMM now than we did twenty years ago, and given its history, many misconceptions abound.

This thesis is therefore presented in the form of a text book: we start from first principles, by defining the basic concepts and terminology, and then build on that base, as we progressively work through the issues associated with proving the scalability of DMM with respect to both CPU and memory.

For readers conversant with the general topic of dynamic memory management, you may wish to skip Chapters 1, 2 and 3, which introduce the subject and cover the general engineering aspects. For those not interested in the history of DMM, you may wish to skip Chapter 4.

Chapter 1 Basic Concepts presents an overview of dynamic memory management sub-systems. We discuss the traditional concepts of allocation, freeing, splitting, coalescing, fragmentation, exhaustion, compaction and expansion.

Chapter 2 Fundamental Techniques supplies a catalog of dynamic memory management techniques. We discuss how such sub-systems can view and track memory using free lists, vectors, trees and bitmaps. Various allocation policies, such as first fit and best fit are presented. Header and footer boundary tags, as well as alignment issues, are also discussed.

Chapter 3 Engineering Issues discusses the time-space tradeoffs of DMM design. This includes issues such as patterned behavior of programs, performance, fragmentation, spatial and temporal locality, scalability, cache effects, robustness and reliability.

Chapter 4 History of the Art reviews the 50+ years of literature on this topic. We observe how the evolution of dynamic memory management is entwined with the evolution of hardware, operating systems, data structures and programming languages. We indicate where and when issues such as fragmentation and patterned behavior appeared and how these issues were subsequently refined.

Chapter 5 State of the Art examines the current state of DMM by investigating and comparing four dynamic memory managers which are currently in widespread use. It is here we observe that no specific work has been done on very large, highly concurrent dynamic memory management.

Chapter 6 Technical Requirements states our requirements for a very large, highly concurrent dynamic memory manager. We define the terms *very large* and *highly concurrent*, going on to argue that while a dynamic memory manager achieves its objectives it should exhibit efficient and predictable, linearly scalable, O(1) time-space behavior.

Chapter 7 Design Discussion explores the solution space via fourteen different thought experiments. We restate our objectives, divide our concerns and derive three principles which guide us to a workable solution.

Chapter 8 Formal Specification presents a formal specification for RHmalloc – a very large, highly concurrent dynamic memory manager. This includes graphical representations of the data structures, as well as pseudo-code for the main algorithms.

Chapter 9 Proving Scalability describes our implementation and testing of RHmalloc, tables our empirical and theoretical results proving the scalability of DMM to indefinitely large, highly concurrent environments.

Chapter 10 Conclusion and Summary states our final conclusions on dynamic memory scalability and discusses possible future work. We also provide a chapter-based summary of this thesis.

Acknowledgements

First and foremost, to Tom Hintz, my supervisor, thank you. Your wealth of knowledge, gentle wisdom, dexterous subtlety and good humor made the entire journey of this doctorate both enlightening and enjoyable. I have much to be grateful for.

I would also like to thank Bruce Howarth for taking the time to review this thesis, and provide valuable feedback; Mary O'Kane for her friendship and moral support; Tony Benson and Gary David for their belief in prodding me into this in the first place; as well as Michael Fry and Barry Jay for their sound advice when I first started.

I am grateful to Linda McEntee for her help in finding some of the rarer papers, and to IBM Australia for use of their resources in the final production of this thesis.

I remain forever appreciative of my undergraduate lecturers: Peter Cheeseman, John Debenham, Jenny Edwards, John Hughes, John Cady, John Colville and Ury Szewcow. The older I get, the more I realize, just how good an education I received.

I have had the privilege of working with a number of outstanding engineers, from whom I have learned so much: Rachna Hariharan, Michael Cahill, Adrian Tullock, Colin Pickup, Ian Steventon, Michael Hollins, Jim Wee, Frank Carnovale, Robbie Gates, Kim Haines, Gary Aitchison, Carlos Enciso, Jason Chan and Paul Smith. I would especially like to mention Alka Yamarti for the many years we have spent coding together.

I am indebted to Kay Hogan and Terry Slack for continuing to help me in countless ways, and I welcome the recent support from John and Lisa Veizades. I would be seriously remiss in failing to acknowledge Rachael Patterson – the person who finally taught me how to write, by sharing with me, her joy in writing. And a sincere thank you to the Krzyszton's and Zinkiewicz's for all the early encouragement.

I am deeply grateful for everything my parents Zygmunt and Maria Huetter have ever done for me; for their love, encouragement and example. All of my family has supported me with love and patience: Sarah, Liz, John, Graham and Brenda. I am particularly thankful to Irene Huetter and Derek Lucas who allowed me to use 'The Cottage' where most of this thesis was written.

And Jenni, the love of my life, above all, thank you for our two beautiful daughters.

Table of Contents

ACKNOWLEDGEMENTS ..... **VII**

1 BASIC CONCEPTS **1**

1.1 INTRODUCTION 1

1.1.1 An Architecture for Toby *3*

1.1.2 Dynamic Memory Management *3*

1.1.3 Allocation *4*

1.1.4 Deallocation *5*

1.1.5 Coalescing *6*

1.1.6 Splitting *7*

1.1.7 Exhaustion *7*

1.1.8 Fragmentation *8*

1.1.9 Compaction *9*

1.1.10 Expansion *10*

1.2 FORMALIZING DYNAMIC MEMORY MANAGEMENT 11

2 FUNDAMENTAL TECHNIQUES **13**

2.1 APPLICABILITY 13

2.1.1 General Purpose *13*

2.1.2 Specific Purpose *13*

2.2 MEMORY VIEW 14

2.2.1 Free Form *14*

2.2.2 Patterned *14*

2.3 MEMORY PARTITIONING 16

2.3.1 Single Pool *16*

2.3.2 Multiple Pools *16*

2.4 POOL CONSTITUTION 17

2.4.1 Mixed Pools *17*

2.4.2 Segregated Pools *17*

2.5 SPACE TRACKING 18

2.5.1 Free List *18*

2.5.2 Vector of Free Lists *19*

2.5.3 Tree of Free Lists *20*

2.5.4 Bitmaps *20*

2.6 LIST ORDERING 21

2.6.1 FIFO *21*

2.6.2 LIFO *22*

2.6.3 Size Ordered *22*

2.6.4 Address Ordered *23*

2.7 ALLOCATION POLICY 24
2.7.1 *First fit* 24
2.7.2 *Next Fit* 24
2.7.3 *Best Fit* 25
2.7.4 *Worst Fit* 26
2.7.5 *Exact Fit* 26
2.8 SPLIT POLICY 27
2.8.1 *Always Split* 27
2.8.2 *Never Split* 28
2.8.3 *Threshold Split* 28
2.9 COALESCE POLICY 28
2.9.1 *Always Coalesce* 28
2.9.2 *Never Coalesce* 29
2.9.3 *Deferred Coalescing* 29
2.10 MEMORY COMPACTION 30
2.10.1 *Compacting* 30
2.10.2 *Non-Compacting* 30
2.11 MEMORY EXTENSION 30
2.11.1 *Preserved* 31
2.11.2 *Carved Up* 31
2.12 MEMORY REDUCTION 32
2.12.1 *Trim* 32
2.12.2 *Never Trim* 32
2.13 BOUNDARY TAGS 32
2.13.1 *Header* 33
2.13.2 *Footer* 33
2.14 BLOCK SIZING 33
2.14.1 *Minimum Size* 34
2.14.2 *Size Increment* 34
2.14.3 *Maximum Size* 34
2.15 CONCURRENCY 34
2.15.1 *Single Threaded* 34
2.15.2 *Multithreaded* 34
2.16 SUMMARY 35

3 ENGINEERING ISSUES 37

3.1 BEHAVIORAL PATTERNS 37
3.1.1 *Various Behaviors* 38
3.2 FRAGMENTATION 38
3.2.1 *External versus Internal Fragmentation* 39

3.2.2 *Fragmentation is Insoluble* 39
3.2.3 *Causes of Fragmentation* 40
3.2.4 *Fragmentation is Manageable* 40
3.3 LOCALITY 40
3.3.1 *Spatial versus Temporal Locality* 41
3.3.2 *The Implications of Locality* 41
3.3.3 *Locality versus Fragmentation* 41
3.4 PERFORMANCE 42
3.4.1 *Block Caching* 42
3.4.2 *Pre-Allocation* 43
3.4.3 *Lookaside Buffers* 43
3.5 ALIGNMENT 43
3.6 OVERHEADS 44
3.7 SCALABILITY 44
3.8 FALSE SHARING 45
3.9 BLOWUP 45
3.10 ROBUSTNESS 45
3.11 EVALUATION 46

4 HISTORY OF THE ART 47

4.1 THE 1950'S AND 60'S 48
4.1.1 *The UNIVAC – Circa 1952* 48
4.1.2 *FORTRAN – Automatic Programming* 49
4.1.3 *Emergence of Symbolic List Processing* 49
4.1.4 *LISP* 51
4.1.5 *Threaded and Knotted Lists* 51
4.1.6 *Automatic Storage Allocation* 52
4.1.7 *ALGOL 60* 53
4.1.8 *Language Support* 53
4.1.9 *Plex: A Shift in Perspective* 53
4.1.10 *Abstract Trees* 54
4.1.11 *A Complete Subsystem* 55
4.1.12 *The List Data Structure* 56
4.1.13 *Multiword List Items* 56
4.1.14 *A Possible Counter Example* 57
4.1.15 *A Fast Storage Allocator* 57
4.1.16 *ALGOL Gets Records* 58
4.1.17 *Still Constrained* 58
4.1.18 *The AED Free Storage Package* 59
4.1.19 *PL/I List Processing* 59

4.1.20 *External v. Internal Fragmentation* 60
4.2 THE 1970'S 60
4.2.1 *Measuring Segment Sizes* 60
4.2.2 *Statistical Properties of the Buddy System* 61
4.2.3 *Virtual Memory* 61
4.2.4 *Optimal Fit Policy* 61
4.2.5 *Bit Tables* 61
4.2.6 *Operating System Research* 62
4.2.7 *Storage Analysis* 63
4.2.8 *Donald E. Knuth* 63
4.2.9 *Fibonacci System* 63
4.2.10 *Bounds of Dynamic Storage Allocation* 64
4.2.11 *Weighted Buddy System* 64
4.2.12 *Fibonacci Buddy Coalescing* 65
4.2.13 *Comparing Best Fit and First Fit* 65
4.2.14 *Influences on Disk Storage Allocation* 66
4.2.15 *Tracing Execution* 66
4.2.16 *Comparing Next Fit* 67
4.2.17 *A More Comprehensive Test* 67
4.2.18 *The Reality of Buddy Systems* 68
4.2.19 *Worst Case Fragmentation* 68
4.2.20 *Anomalies in Knuth's Rule* 69
4.2.21 *Standard Interface* 69
4.2.22 *Compacting Lists* 69
4.3 THE 1980'S 70
4.3.1 *BSD Unix* 70
4.3.2 *Fragmentation Analysis* 70
4.3.3 *Parallel Memory Allocation* 71
4.3.4 *How Optimal is Optimal Fit?* 71
4.3.5 *Release Match Policy* 72
4.3.6 *Adaptive Lists* 72
4.3.7 *Fast Fits* 73
4.3.8 *SUN OS* 73
4.3.9 *UNIX System V Release 4* 74
4.3.10 *Choosing a new Algorithm for VM/SP* 74
4.3.11 *Two More Buddy Variants* 75
4.3.12 *The Software Lookaside Buffer* 75
4.3.13 *Using First Fit for Tape Archival* 76
4.3.14 *Adaptive List Sizes* 76
4.3.15 *Finer Compacting Algorithms* 77

4.3.16 *Dual Buddy System* ... 77
4.3.17 *More Efficient First Fit* ... 78
4.3.18 *Doug Lea's Allocator* ... 79
4.4 THE 1990'S ... 80
4.4.1 *Fast Allocation* ... 80
4.4.2 *A Persistent Heap* ... 81
4.4.3 *Lifetime Prediction* ... 81
4.4.4 *Hardware Assistance* ... 82
4.4.5 *Cache Locality* ... 83
4.4.6 *Allocation Costs* ... 83
4.4.7 *Parallel Allocation* ... 84
4.4.8 *Synthesizing Custom Allocators* ... 85
4.4.9 *Model Evaluation* ... 86
4.4.10 *GNU C* ... 87
4.4.11 *Customizable C++ Framework* ... 87
4.4.12 *Critical Review* ... 87
4.4.13 *Heap Profiling* ... 89
4.4.14 *Heap Scalability* ... 90
4.4.15 *Vmalloc* ... 91
4.4.16 *An Exhaustive Study* ... 91
4.4.17 *Long-Running Systems* ... 93
4.4.18 *Fragmentation Solved?* ... 96
4.4.19 *Scalable SMP Allocator* ... 96
4.4.20 *PTmalloc* ... 97
4.5 THE 2000'S ... 97
4.5.1 *Hoard* ... 97
4.5.2 *Avon* ... 99
4.5.3 *Debugging Tools* ... 100
4.5.4 *A CPU Scalable Lock-Free Allocator* ... 100
4.6 A REFLECTION ON THE LITERATURE ... 101

5 STATE OF THE ART ... 102

5.1 INTRODUCTION ... 102
5.2 REPRESENTATIVE SET ... 102
5.3 BERGER'S TAXONOMY ... 103
5.4 DISSECTING DLMALLOC ... 106
5.4.1 *Larger Sizes* ... 107
5.4.2 *Deferred Coalescing* ... 109
5.4.3 *CPU Scalability* ... 110
5.4.4 *Memory Scalability* ... 110

5.5 DISSECTING PTMALLOC 111
5.5.1 CPU Scalability *112*
5.5.2 Memory Scalability *112*
5.6 DISSECTING LKMALLOC 112
5.6.1 CPU Scalability *113*
5.6.2 Memory Scalability *114*
5.7 DISSECTING HOARD 114
5.7.1 CPU Scalability *116*
5.7.2 Memory Scalability *116*
5.8 SUMMARY 117

6 TECHNICAL REQUIREMENTS 118

6.1 DISCUSSION 118
6.2 DEFINITION OF TERMS 119
6.3 A SENSE OF SCALE 119
6.4 THE LIMITS OF SCALABILITY 121
6.5 FEASIBILITY 121
6.6 PRACTICAL CONSIDERATIONS 122
6.6.1 Time Efficiency *122*
6.6.2 Space Efficiency *123*
6.6.3 Time v. Space Tradeoff *123*
6.6.4 Distribution Independence *124*
6.6.5 Long-Term Stability *124*
6.7 REQUIREMENTS 125
6.8 DEFINITION 126

7 DESIGN DISCUSSION 127

7.1 INTRODUCTION 127
7.2 OUR PHILOSOPHY 130
7.3 BASIC CONCERNS 131
7.4 GUIDING PRINCIPLES 132
7.5 THOUGHT EXPERIMENT #1 132
7.6 AN OBSERVATION ON DELAYED COALESCING 133
7.7 THOUGHT EXPERIMENT #2 134
7.8 THOUGHT EXPERIMENT #3 135
7.9 THOUGHT EXPERIMENT #4 137
7.10 THOUGHT EXPERIMENT #5 138
7.11 INTERIM CONCLUSION #1 139
7.12 PUSHING FORWARD 140
7.13 THOUGHT EXPERIMENT #6 141

7.14 A Third Guiding Principle Emerges 142
7.15 Thought Experiment #7 142
7.16 Thought Experiment #8 143
7.17 Interim Conclusion #2 144
7.18 Thought Experiment #9 144
7.19 Thought Experiment #10 145
7.20 A Valuable Insight 147
7.21 Thought Experiment #11 147
7.22 Critical Insight #1 148
7.23 Using Trees 148
7.24 Using Vectors 149
7.25 Thought Experiment #12 149
7.26 Thought Experiment #13 150
7.27 Introducing Bit Indexes 151
7.28 Bit Index Optimization 152
7.29 Bit Indexes and Concurrency 152
7.30 Thought Experiment #14 153
7.30.1 Managing a List of Sub-Heaps 153
7.30.2 Detecting Excessive Collisions 154
7.30.3 Shifting Sub-Heaps between Lists 155
7.30.4 Releasing Sub-Heaps 155
7.30.5 Cross-Check: Cache Efficiency 155
7.30.6 Visualizing RHmalloc on an SMP Machine 159
7.31 Summary: A First Order Sketch 162

8 FORMAL SPECIFICATION 164

8.1 Data Structures 164
8.1.1 The Sub-Heap 164
8.1.2 The Heap 171
8.2 Algorithms 178
8.2.1 Allocate Block 178
8.2.2 Free Block 179
8.2.3 Pop Chunk 180
8.2.4 Push Chunk 181
8.2.5 Split Chunk 182
8.2.6 Coalesce Chunks 183
8.2.7 Find Sub-heap 184
8.2.8 Shift Sub-heap 185
8.2.9 Create Sub-heap 186
8.2.10 Destroy Sub-heap 187

8.2.11 Initialize Heap *188*

9 PROVING SCALABILITY **189**

9.1 DISCUSSION 190
9.2 PROPOSITION 190
9.3 REVIEWING RHMALLOC 191
9.4 USING A 32-BIT MACHINE 191
9.5 1PB BY EXTRAPOLATION 192
9.6 THEORETICALLY INDEFINITE 192
9.7 TEST SUITE 192
9.8 INSTRUMENTATION 193
9.9 NOTES 195
9.10 SIDE EFFECTS 196
9.11 TEST 1 – WITHIN SUB-HEAP RESPONSE TIME 197
9.11.1 Objective *197*
9.11.2 Description *197*
9.11.3 Results *197*
9.11.4 Comments *198*
9.11.5 Conclusion *198*
9.12 TEST 2 – FIND SUITABLE SUB-HEAP TIME 199
9.12.1 Objective *199*
9.12.2 Description *199*
9.12.3 Results *200*
9.12.4 Comments *201*
9.12.5 Conclusion *202*
9.13 TEST 3 – OVERALL RESPONSE TIME 203
9.13.1 Objective *203*
9.13.2 Description *203*
9.13.3 Results *203*
9.13.4 Comments *208*
9.13.5 Conclusion *209*
9.14 A MORE DETAILED LOOK 210
9.14.1 Group A – Single Sub-Heap *211*
9.14.2 Group B – Independent Working Sets *211*
9.14.3 Group C – Interwoven Working Sets *212*
9.14.4 The Three Sub-Curves *212*
9.14.5 Long Running Steady State Systems *213*
9.14.6 Block Coalescing *214*
9.14.7 Cache Related Effects *215*
9.14.8 A 1PB Heap on a 1,024 CPU Machine *216*

9.14.9 The Inflection Points *217*
9.14.10 Possible Inherent Imbalance *218*
9.14.11 Malloc - Free State Transition Diagram *219*
9.14.12 A Fourth Maxim of Scalability *220*
9.14.13 A Comment on Entropy *221*
9.14.14 Entropy, Cache and the Cost to Free *221*
9.14.15 Systematic v. Stochastic Processes *222*
9.15 A Final Comment on Our Testing Approach 226
9.16 Overall Conclusions 227

10 CONCLUSION AND SUMMARY 228

10.1 Overall Perspective 228
10.1.1 Heaps are Provably Scalable *229*
10.1.2 Why are Heaps Scalable? *229*
10.1.3 Why is any Scalable System Scalable? *231*
10.1.4 Four Maxims of Scalability *231*
10.2 Future Directions 232
10.2.1 Heaps in General *232*
10.2.2 Scalability in General *233*
10.3 Conclusion 234
10.4 Summary 235
10.4.1 Chapter 1 – Introduction *235*
10.4.2 Chapter 2 – Basic Techniques *235*
10.4.3 Chapter 3 – Engineering Issues *236*
10.4.4 Chapter 4 – History of the Art *238*
10.4.5 Chapter 5 – State of the Art *239*
10.4.6 Chapter 6 – Technical Issues *240*
10.4.7 Chapter 7 – Design Discussion *240*
10.4.8 Chapter 8 – Formal Specification *242*
10.4.9 Chapter 9 – Proving Scalability *243*
10.4.10 Chapter 10 – Summary and Conclusion *243*

APPENDIX A – NOTATION AND CONVENTIONS 244

APPENDIX B – REPORTED HEAP SIZES 245

APPENDIX C – TIMELINE OF DYNAMIC MEMORY MANAGEMENT 246

APPENDIX D – HUETTER'S DMM TAXONOMY 247

APPENDIX E – FULL DEPICTION OF RHMALLOC 249

GLOSSARY 250

BIBLIOGRAPHY 257

List of Figures

FIGURE 1.1: VISUALIZING TOBY'S MEMORY POOL AS A 10 × 10 GRID. 1
FIGURE 1.2: TOBY'S MEMORY POOL WITH INITIAL SET OF WORDS – WITH DOG HIGHLIGHTED. 2
FIGURE 1.3: A HIGH-LEVEL ARCHITECTURE FOR TOBY. 3
FIGURE 1.4: TOBY'S MEMORY POOL WITH SCHOOL ADDED. 4
FIGURE 1.5: TOBY'S MEMORY POOL AFTER MONSTER IS DELETED. 5
FIGURE 1.6: COALESCING TWO FREE BLOCKS INTO A SINGLE BLOCK. 6
FIGURE 1.7: SPLITTING A FREE BLOCK. 7
FIGURE 1.8: FRAGMENTED MEMORY POOL. 8
FIGURE 1.9: COMPACTING A MEMORY POOL. 9
FIGURE 1.10: EXPANDING A MEMORY POOL 10
FIGURE 1.11: THE PAGE VIEW OF VIRTUAL MEMORY VERSUS THE BLOCK VIEW OF DYNAMIC MEMORY. 11
FIGURE 2.1: AN EXAMPLE OF A FREE FORM VIEW OF MEMORY 14
FIGURE 2.2: AN EXAMPLE OF A HIERARCHICALLY PATTERNED VIEW OF MEMORY. 15
FIGURE 2.3: AN EXAMPLE OF A PARTITIONED HEAP WITH TWO POOLS. 16
FIGURE 2.4: EXAMPLE OF A HEAP WITH TWO MIXED POOLS. 17
FIGURE 2.5: AN EXAMPLE OF A HEAP WITH TWO SEGREGATED POOLS. 18
FIGURE 2.6: EXAMPLE OF A FREE LIST WITH FOUR BLOCKS OF SIZES 4, 2, 3 AND 1 RESPECTIVELY. 18
FIGURE 2.7: AN EXAMPLE OF A FOUR ELEMENT VECTOR WITH THREE FREE LISTS. 19
FIGURE 2.8: AN EXAMPLE OF A BINARY TREE OF THREE FREE LISTS. 20
FIGURE 2.9: AN EXAMPLE OF A BITMAP USED TO TRACK FREE SPACE IN A MEMORY POOL. 21
FIGURE 2.10: A FIFO ORDERED FREE LIST SHOWING A BLOCK BEING ADDED AT THE TAIL. 21
FIGURE 2.11: A LIFO ORDERED FREE LIST SHOWING A BLOCK BEING ADDED AT THE HEAD. 22
FIGURE 2.12: A SIZE ORDERED FREE LIST SHOWING A BLOCK BEING INSERTED MIDWAY. 22
FIGURE 2.13: AN ADDRESS ORDERED FREE LIST WITH A BLOCK BEING INSERTED BY ADDRESS. 23
FIGURE 2.14: FIRST FIT ALLOCATION POLICY SHOWING THE LIST BEING SEARCHED FROM THE HEAD, FOR THE FIRST BLOCK WHICH SATISFIES THE REQUEST. 24
FIGURE 2.15: NEXT FIT ALLOCATION POLICY SHOWING THE LIST BEING SEARCHED FROM WHERE THE LAST SEARCH ENDED, LOOKING FOR THE NEXT BLOCK WHICH SATISFIES THE REQUEST. 25
FIGURE 2.16: BEST FIT ALLOCATION POLICY SHOWING THE SELECTION OF THE CLOSEST SIZE WHICH SATISFIES THE REQUEST. 25
FIGURE 2.17: WORST FIT ALLOCATION POLICY SHOWING THE LARGEST BLOCK IS SELECTED AND THEN CARVED UP (REMAINDER BEING RETURNED TO THE LIST). 26
FIGURE 2.18: EXACT FIT ALLOCATION POLICY SHOWING NO SUITABLE BLOCK WAS FOUND SO MEMORY WAS EXTENDED (AND THEN CARVED UP) TO SATISFY THE REQUEST. 27
FIGURE 2.19: ALWAYS SPLIT POLICY. 27
FIGURE 2.20: NEVER SPLIT POLICY. 28
FIGURE 2.21: ALWAYS COALESCE POLICY MERGES ADJACENT FREE BLOCKS WHEN THEY ARE FREED. 29
FIGURE 2.22: NEVER COALESCE POLICY PERMITS ADJACENT FREE BLOCKS. 29

FIGURE 2.23: COMPACTING A MEMORY POOL SHUFFLES ALLOCATED BLOCKS TOGETHER TO COMBINE FREE SPACE. 30
FIGURE 2.24: PRESERVING THE MEMORY EXTENSION BY ADDING IT AS ONE LARGE FREE BLOCK. 31
FIGURE 2.25: CARVING UP THE MEMORY EXTENSION INTO A COMMONLY USED SIZE. 32
FIGURE 2.26: A BLOCK HEADER. 33
FIGURE 2.27: A BLOCK FOOTER. 33
FIGURE 5.1: SERIAL SINGLE HEAP. FIGURE SHOWS ONE THREAD ACCESSING THE HEAP, WITH THREE OTHER THREADS WAITING. 103
FIGURE 5.2: CONCURRENT SINGLE HEAP. FIGURE SHOWS THREE THREADS WHICH ARE CONCURRENTLY ACCESSING THE SAME HEAP. 104
FIGURE 5.3: PURE PRIVATE SUB-HEAPS. WHERE THREAD OWNING PRIVATE SUB-HEAP A ALLOCATES A BLOCK WHICH IS THEN FREED BY THREAD OWNING PRIVATE SUB-HEAP B. BLOCK IS RETURNED TO PRIVATE SUB-HEAP B. 105
FIGURE 5.4: PRIVATE SUB-HEAPS WITH OWNERSHIP. WHERE THE MEMORY BLOCK ALLOCATED BY THREAD OF PRIVATE SUB-HEAP A AND THEN FREED BY THREAD OF SUB-HEAP B IS RETURNED TO THE ORIGINAL SUB-HEAP I.E. PRIVATE SUB-HEAP A. 105
FIGURE 5.5: PRIVATE SUB-HEAPS WITH THRESHOLDS. FIGURE SHOWS A MICRO-HEAP BEING SHIFTED FROM PRIVATE SUB-HEAP B INTO THE COMMON SUB-HEAP. 106
FIGURE 5.6: DEPICTING THE EXACT BINS IN DLMALLOC AS A VECTOR OF LISTS. 107
FIGURE 5.7: DEPICTING THE SORTED BINS IN DLMALLOC AS A VECTOR OF LISTS. 108
FIGURE 5.8: FAST BINS ARE DEFERRED COALESCING BINS MIRRORING THE NORMAL BINS. 110
FIGURE 5.9: PTMALLOC IS A LIST OF GUARDED DLMALLOC INSTANCES. 111
FIGURE 5.10: LKMALLOC IS A SET OF PRIVATE SUB-HEAPS WITH OWNERSHIP. 113
FIGURE 5.11: HOARD IS AS A SET OF PRIVATE SUB-HEAPS CONSISTING OF A SET OF SUPERBLOCKS. SUPERBLOCKS CAN MOVE BETWEEN PRIVATE SUB-HEAPS AND A COMMON SUB-HEAP. 115
FIGURE 7.1: A SIMPLE POWER-OF-TWO ALLOCATOR. 128
FIGURE 7.2: ABSTRACT MODEL OF AN INDEFINITELY SCALABLE SMP MACHINE. THE BORDERS OF THE CIRCLES MAY BE THOUGHT OF AS BUSES. 130
FIGURE 7.3: TEX-1 – A SINGLE LINKED LIST OF FREE BLOCKS GUARDED BY LOCK L. 132
FIGURE 7.4: TEX-2 – A VECTOR OF GUARDED LISTS. 134
FIGURE 7.5: TEX-3 – A VECTOR OF GUARDED EXACT LISTS AND A VECTOR OF GUARDED SORTED LISTS... 135
FIGURE 7.6: TEX-4 – A VECTOR OF GUARDED SMALL SIZE LISTS AND A VECTOR OF GUARDED LARGE SIZE LISTS. 137
FIGURE 7.7: TEX-5 – A VECTOR OF GUARDED LISTS EXPLICITLY COVERING ALL POSSIBLE SIZES. 138
FIGURE 7.8: TEX-6 – ONE PRIVATE SUB-HEAP PER THREAD. 141
FIGURE 7.9: TEX-7 – ONE GUARDED SUB-HEAP PER THREAD, FREED BLOCKS BEING ADDED TO THE SUB-HEAP ASSOCIATED WITH FREEING THREAD. 142
FIGURE 7.10: TEX-8 – ONE GUARDED SUB-HEAP PER THREAD, FREED BLOCKS BEING RETURNED TO THE SUB-HEAP FROM WHICH THEY WERE ALLOCATED. 143

FIGURE 7.11: TEX-9 – COMMON SUB-HEAP ACCESSIBLE BY ALL THREADS, PLUS A SUB-HEAP PRIVATE TO EACH THREAD. 144
FIGURE 7.12: TEX-10 – MULTIPLE COMMON SUB-HEAPS ACCESSIBLE BY ALL THREADS, PLUS A SUB-HEAP PRIVATE TO EACH THREAD. 146
FIGURE 7.13: TEX-11 – A HEAP CONSISTING OF A SET OF COMMON SUB-HEAPS. 147
FIGURE 7.14: TEX-12 – TRACKING SUB-HEAPS WITH TWO VECTORS – THE SMALLER ONE IS EXACT, WHILE THE OTHER IS LOGARITHMICALLY SPACED. 150
FIGURE 7.15: TEX-13 – A SINGLE VECTOR OF SUB-HEAPS. 151
FIGURE 7.16: TEX-14 – A VECTOR OF SUB-HEAPS WITH A TWO LEVEL BIT INDEX. 153
FIGURE 7.17: FIGURE DEPICTS A THREAD SELECTING A SUB-HEAP OF SIZE TWO AND SHIFTING IT TO THE END OF THAT LIST (REFER DASHED LINE). 154
FIGURE 7.18: FIGURE DEPICTS A THREAD SHIFTING A SUB-HEAP FROM LIST 2 TO LIST 1 (REFER DASHED LINE). 155
FIGURE 7.19: A STYLIZED DEPICTION OF HOW RHMALLOC MAPS OVER AN SMP MACHINE. 160
FIGURE 7.20: A FIRST-ORDER SKETCH OF RHMALLOC. 162
FIGURE 8.1: HEAP CONTROL BLOCK MANAGES THE SET OF SUB-HEAPS. 164
FIGURE 8.2: SUB-HEAPS MAP ONTO NON-OVERLAPPING VIRTUAL MEMORY RANGES. 165
FIGURE 8.3: AN UNBOUNDED HEAP VIEWED AS AN INDEFINITE SET OF FINITE SUB-HEAPS. 166
FIGURE 8.4: SUB-HEAPS ARE A STRUCTURE CONTAINING INFORMATION USED TO CONTROL ALLOCATION FROM THE REMAINING SPACE IN THE STRUCTURE. 167
FIGURE 8.5: THE FREE LIST VECTOR 167
FIGURE 8.6: THE FREE BITMAP. 168
FIGURE 8.7: THE FREE LOCK VECTOR 169
FIGURE 8.8: BANDED SIZE VERSUS EXACT SIZE FREE LISTS. 170
FIGURE 8.9: THE RHMALLOC SUB-HEAP (BLOCK SIZES NOT DRAWN TO SCALE). 171
FIGURE 8.10: SUB-HEAPS ORGANIZED BY AVAILABLE SPACE 172
FIGURE 8.11: THE SPACE BITMAP. 173
FIGURE 8.12: THE SPACE LOCK VECTOR 174
FIGURE 8.13: ROTATING A SPACE LIST. 175
FIGURE 8.14: SHIFTING A SUB-HEAP WITHIN THE SPACE LISTS. 176
FIGURE 8.15: DEMOTING AND PROMOTING A SUB-HEAP WITHIN THE SPACE LIST VECTOR. 177
FIGURE 9.1: TEST 1 RESULTS SHOWING AVERAGE WITHIN SUB-HEAP RESPONSE TIME **R** PER REQUEST (IN MICROSECONDS) 197
FIGURE 9.2: TEST 2 RESULTS SHOWING THE AVERAGE TIME **F** TO FIND A SUITABLE SUB-HEAP TO SATISFY AN ALLOCATION REQUEST. 200
FIGURE 9.3: TEST 3 RESULTS SHOWING THE AVERAGE TIME PER REQUEST IN MICROSECONDS FOR EACH RUN. 203
FIGURE 9.4: TEST 3 RESULTS SHOWING THE PERCENTAGE OF REQUESTS WHICH REQUIRE SUB-HEAP SHIFTS (SF) AND THE PERCENTAGE OF REQUESTS WHICH REQUIRE SUB-HEAP FINDS (FS) FOR EACH RUN. . 204

FIGURE 9.5: TEST 3 RESULTS SHOWING THE PERCENTAGE OF REQUESTS WHICH INCUR SUB-HEAP CONTENTION FOR EACH RUN. 205
FIGURE 9.6: TEST 3 RESULTS SHOWING THE AVERAGE TIME PER SUB-HEAP SHIFT (IN MICROSECONDS) FOR EACH RUN. 206
FIGURE 9.7: TEST 3 RESULTS THE TIME PER SUB-HEAP FIND (IN MICROSECONDS) FOR EACH RUN. 207
FIGURE 9.8: THE 256MB HEAP RESPONSE PATTERN – RECTANGLES SHOW SUB-HEAP SIZE. 210
FIGURE 9.9: PERCENTAGE OF REQUESTS THAT INVOLVE SUB-HEAP FINDS FOR THE 256MB HEAP. 213
FIGURE 9.10: NUMBER OF ALLOCATIONS AND FREES FOR THE 256MB HEAP. 214
FIGURE 9.11: PERCENTAGE OF FREES WHICH COALESCE BLOCKS FOR THE 256MB HEAP. 214
FIGURE 9.12: TIME TO ALLOCATE A BLOCK VERSUS TIME TO FREE FOR 256MB HEAP. 217
FIGURE 9.13: TIME SPENT SPLITTING BLOCKS VERSUS TIME SPENT COALESCING BLOCKS FOR 256MB HEAP. 218
FIGURE 9.14: MALLOC – FREE STATE TRANSITION DIAGRAM. 219
FIGURE 9.15: A HEAP IN A DYNAMIC STEADY STATE. 223
FIGURE 9.16: THE CONTINUOUS INTERACTION BETWEEN A HEAP AND A MULTITHREADED SYSTEM. 223
FIGURE 10.1: CPU AND MEMORY PLOT INDICATING RELATIVE SIZE OF VARIOUS ACTIVITIES. 228
FIGURE 10.2: A HEAP IS SIMULTANEOUSLY DIVISIBLE AND EXPANDABLE. 230

List of Tables

TABLE 2.1: SUMMARY OF FUNDAMENTAL DMM ISSUES AND TECHNIQUES. 36
TABLE 4.1: THE BETTER ALLOCATORS FROM JOHNSTONE'S FRAGMENTATION AND LOCALITY STUDY..... 92
TABLE 4.2: THE WORST ALLOCATORS FROM JOHNSTONE'S FRAGMENTATION AND LOCALITY STUDY. 92
TABLE 6.1: TIME (IN SECONDS) TO ALLOCATE ALL OF MEMORY JUST ONCE GIVEN CPU SPEED AND MEMORY CAPACITY – ASSUMING AVERAGE BLOCK SIZE OF 100 BYTES AND 1,000 INSTRUCTIONS TO ALLOCATE A BLOCK. 120
TABLE 9.1: TABLE OF STATISTICS COLLECTED IN RHMALLOC TEST RUNS. 195
TABLE 9.2: TIME TO PERFORM VARIOUS OPERATIONS IN TEST 1. 198
TABLE 9.3: TIME TO PERFORM VARIOUS OPERATIONS IN TEST 2. 201
TABLE 9.4: TIME TO PERFORM VARIOUS OPERATIONS IN TEST 3. 207